2000

Integer multiplication with overflow detection or saturation

Mustafa Gok
Lehigh University

Gok, Mustafa

Integer Multiplication with Overflow Detection or Saturation

June 2000

INTEGER MULTIPLICATION WITH OVERFLOW DETECTION OR SATURATION

by

Mustafa Gok

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Electrical Engineering

Lehigh University

May 2000

This thesis is accepted in partial fulfillment of the requirements for the degree of Master of Science.

May 4, 2000
(Date)

Prof. Michael J. Schulte
(Thesis Advisor)

Prof. Bruce D. Fritchman
(Chairperson of Department)

Acknowledgments

I would like to thank my advisor Prof. Michael J. Schulte, for his encouragement, guidance and friendship. He has always been willing to help and correct my mistakes. I have learned so much from him not only related to this thesis but beyond the scope of the thesis. Thanks for everything that I cannot mention in this limited text. Finally, I would like to thank my parents, Hülya and Şener for their love, care and support.

Table of Contents

List of Tables

List of Figures

Abstract

High-speed multipliers typically produce products that have the same length as their input operands, even though the true product may be twice as long as the inputs. If the product of an integer multiply exceeds the number of bits allocated for the result, overflow occurs and needs to be detected. On some processors, results that overflow saturate to the most positive or most negative representable number.

This thesis presents new methods for overflow detection or saturation for unsigned and two's complement parallel multipliers. These methods significantly reduce the area and delay of the parallel multipliers without effecting the regularity of their structures. Synthesis results show area reductions for unsigned parallel multipliers between 47% and 53% and area reductions for two's complement parallel multipliers between 35% and 47%. The delays of the parallel multipliers are reduced 23% to 42% for unsigned multipliers and 24% to 48% for two's complement multipliers.

Chapter 1

Introduction

1.1 Multiplication

Multiplication is an essential arithmetic operation for general purpose computers and digital signal processors. High-performance systems support parallel multiplication in hardware. Various high-speed parallel multipliers have been proposed and realized. Most parallel multiplier designs can be divided into two classes; array multipliers and tree multipliers. Array multipliers consist of an array of similar cells that generate and accumulate the partial products [3]. Tree multipliers generate all partial products in parallel, use a tree of counters to reduce the partial products to sum and carry vectors and then sum these vectors, using a fast carry-propagate adder. Several methods have been developed for reducing the partial products [1], [2], [4].

The regular structure of array multipliers facilitates their implementation in VLSI technology. The delay of array multipliers, however, is proportional to the operand length. On the other hand, tree multipliers offer a delay proportional to the logarithm of the operand length. The main drawback of tree multipliers is their irregular interconnection structure, which makes them difficult to implement in VLSI. Thus, tree multipliers are preferred for high performance systems, while array multipliers are preferred for systems requiring less area. New implementations and optimizations of parallel multipliers are still active research areas [5]. More detail descriptions of array and tree multipliers are given in the next chapter.

1.2 Overflow

To avoid grow in word length, most instruction set architectures and high-level languages require that arithmetic operation return results with the same length as their input operands. If the result of an integer arithmetic operation on n-bit numbers cannot be represented by n bits, overflow occurs and needs to be detected. For integer multiplication, the method for overflow detection also depends on whether the operands are signed or unsigned integers. For unsigned multiplication, overflow only occurs if the result is larger than the largest unsigned n-bit number. For signed integer multiplication overflow also occur if the result is smaller than the minimum representable n-bit number. For two's complement multiplication there is also a

difference between fractional and integer overflow detection. Since overflow only occurs for two's complement fractional numbers when -1 is multiplied by -1, it is easy to detect overflow when multiplying two's complement fractions.

It is an important design issue in computer architecture is to decide what to do when overflow occurs. Typically, overflow results in an overflow flag being set. This overflow flag can then be used to signal an arithmetic exception [9].

1.3 Saturation

In most general purpose processors, overflow is handled by setting an exception flag. More recent implementations for digital signal processing and multimedia applications saturate results that overflow to the most positive or most negative representable number [11], [12]. For two's complement integers this is -2^{n-1} for negative numbers and $2^{n-1}-1$ for positive numbers. For unsigned integers, results that are too large saturate to 2^n-1.

1.4 Thesis Overview

Previous studies have focussed on overflow detection in two's complement addition [13], multi-operand addition [14], fractional arithmetic operations [10] and generalized signed-digit arithmetic[15]. This thesis presents efficient techniques for integer multiplication with overflow detection or saturation. Most existing computers detect

overflow in integer multiplication by a computing $2n$-bit product and then testing the most significant bits to see if overflow has occurred. The methods proposed in this thesis only calculate n or $n+1$ bits of the product. This leads to a significant reduction in area and delay.

Chapter 2 presents array multipliers, tree multipliers, and conventional methods for overflow detection. Chapter 3 introduces new methods for overflow detection and saturation for unsigned integer multiplication. Chapter 4 focuses on overflow detection and saturation for two's complement integer multiplication. Chapter 5 presents the component counts and area and delay estimates for unsigned and two's complement parallel multipliers that use either the conventional or the proposed methods for overflow detection. Chapter 6 discusses future work and gives conclusions.

Chapter 2

Previous Research

2.1 Unsigned Parallel Multipliers

Multiplication of two n-bit unsigned numbers is shown in Figure 2.1, where

$$\begin{aligned} A &= a_{n-1}a_{n-2}a_{n-3}\ldots a_1a_0 \quad \text{(Multiplicand)} \\ B &= b_{n-1}b_{n-2}b_{n-3}\ldots b_1b_0 \quad \text{(Multiplier)} \\ P &= p_{2n-1}p_{2n-2}p_{2n-3}\ldots p_1p_0 \quad \text{(Product)} \end{aligned} \tag{2.1}$$

Multiplication produces a $2n$-bit product. If the n least significant bits $p_{n-1}\ldots p_0$ are used for the result, then overflow occurs when the actual product uses more than n bits. In other words, overflow occurs when the product is greater or equal to 2^n.

With conventional methods, overflow is detected after the $2n$-bit result is produced. This is done by simply logically ORing together the n most significant bits,

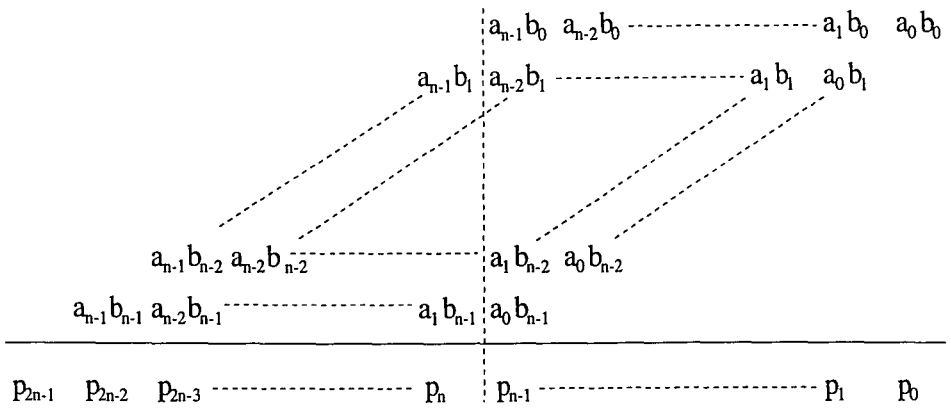

Figure 2.1: Multiplication of A and B.

such that

$$V = p_{2n-1} + p_{2n-2} \ldots p_{n+1} + p_n \tag{2.2}$$

where V is one if overflow occurs and + denotes logical OR. Although calculating $2n$ product bits and then detecting overflow leads to unnecessary area and delay, most computers that provide integer multiplication with overflow detection use this approach. If the system requires saturation, the saturated least significant product bits $\langle p \rangle = \langle p_{n-1} \rangle \langle p_{n-2} \rangle \ldots \langle p_0 \rangle$ are computed as

$$< p_i >= p_i + V \qquad 0 \leq i \leq n-1 \tag{2.3}$$

which sets the product to 2^{n-1} when overflow occurs.

2.1.1 Unsigned Array Multipliers

In general, array multipliers are slower than tree multipliers. In spite of this speed disadvantage, however, array multipliers are often used due their regular layout, low

area, and simplified design.

A block diagram of an unsigned 8 by 8 array multiplier is shown in Figure 2.2. Each diagonal in Figure 2.2 corresponds to a column in the multiplication matrix, in Figure 2.1. A modified half adder (MHA) is a half adder with an AND gate, and a modified full adder (MFA) is a full adder with an AND gate. The AND gates generate the partial products. Full adders and half adders add the generated partial products. Sum outputs are connected diagonally and carry outputs are connected vertically. The last row of adders, which are connected from left to right, generates the n-most significant product bits . The critical path through this multiplier is shown with dashed lines. Since almost half of the latency is due to the bottom row of adders, this row may be replaced by a fast carry-propagate adder. Although this decreases the overall delay, it has a negative impact on the design's regularity. An n by n unsigned array multiplier uses n^2 AND gates, $(n^2 - 2n)$ FAs and n HAs.

The conventional method for overflow detection requires the n most significant product bits to be calculated. These product bits are then OR'ed together to produce the overflow flag, as shown in Figure 2.2. The conventional method for saturating multiplication is accomplished by ORing V with p_0 to p_{n-1}, as shown in Figure 2.3.

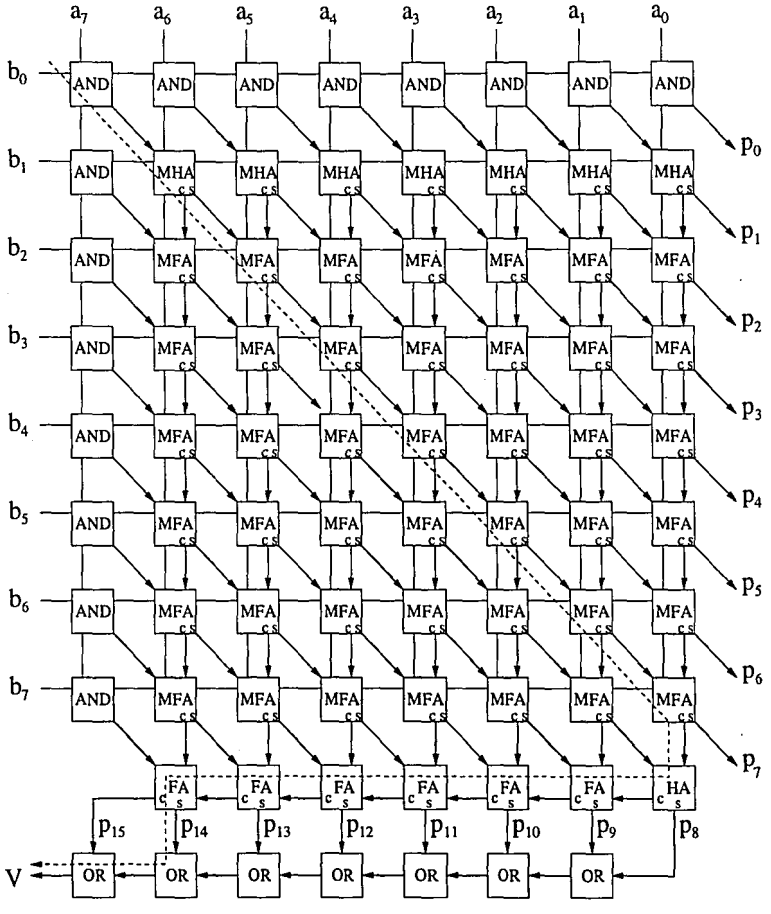

Figure 2.2: Unsigned Array Multiplier with Conventional Overflow Detection.

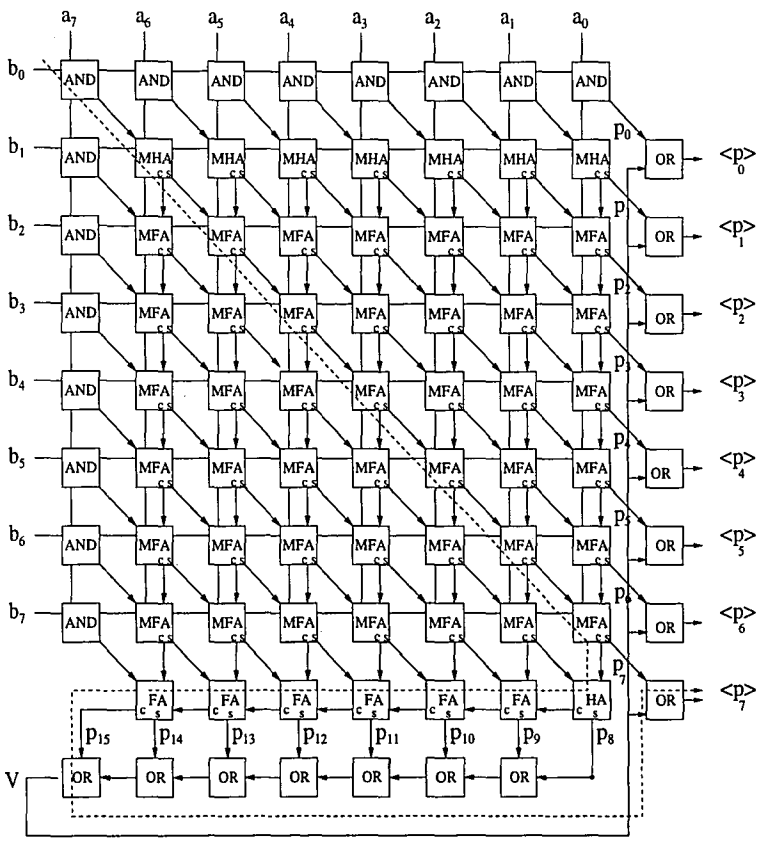

Figure 2.3: Unsigned Array Multiplier with Conventional Saturation.

2.1.2 Unsigned Tree Multipliers

Tree multipliers have three main parts; partial product generation, partial product reduction, and final carry-propagate addition. Various reduction schemes have been developed over the years. Two of the most well-known methods for multiplier tree designs are those proposed by Wallace [2] and Dadda [1]. Wallace's strategy combines three rows of partial product bits using $(3,2)$ and $(2,2)$ counters to produce two rows. Dadda's strategy leads to a simpler counter tree but requires a larger final carry-propagate adder. There are hybrid approaches between these two methods that offers cost and speed trade-offs for VLSI implementations. These reduction schemes differs in the number and placement of counters in the tree and the size of the final carry propagate adder. The tree multipliers presented in this thesis use Dadda's method, since it allows component counts to easily be determined based on n. Since our overflow detection method does not depend on the reduction strategy, similar savings are expected for other tree multipliers.

Figure 2.4 shows a dot diagram of an 8 by 8 unsigned array multiplier. Dot diagrams are often used to illustrate reduction strategies in tree multipliers [2]. With this technique, a dot represents a partial product bit, a plain diagonal line represents a full adder and a crossed diagonal line represents a half adder. The two bottom rows of the dot diagram corresponds to sum and carry vectors that are combined using the final carry-propagate adder to produce the product.

Dadda multipliers require n^2 AND gates, $(n^2 - 4n + 3)$ FAs, $(n-1)$ HAs and a

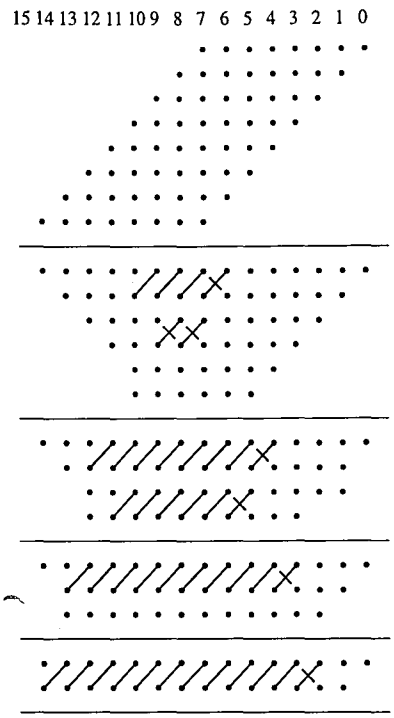

Figure 2.4: Dadda Reduction Scheme.

$(2n-2)$-bit CPA. The number of stages, based on n, is shown in Table 2.1.

For example a 24 by 24 Dadda multiplier requires seven reduction stages. The worst case delay path is equal to the delay of partial product generation, plus the delay for the reduction stages, plus the delay for the final carry-propagate addition.

With the conventional method, overflow is detected with a tree of $(n-1)$ 2-input OR gates. The delay of the tree of OR gates is equivalent to $\lceil \log_2 n \rceil$ 2-input OR gates, as shown in Figure 2.5.

Range of n	s
$3 \leq n \leq 3$	1
$4 \leq n \leq 4$	2
$5 \leq n \leq 6$	3
$7 \leq n \leq 9$	4
$10 \leq n \leq 13$	5
$14 \leq n \leq 19$	6
$20 \leq n \leq 28$	7
$29 \leq n \leq 42$	8
$43 \leq n \leq 63$	9
$64 \leq n \leq 94$	10

Table 2.1: Number of Stages s for n-Bit Dadda Tree Multipliers.

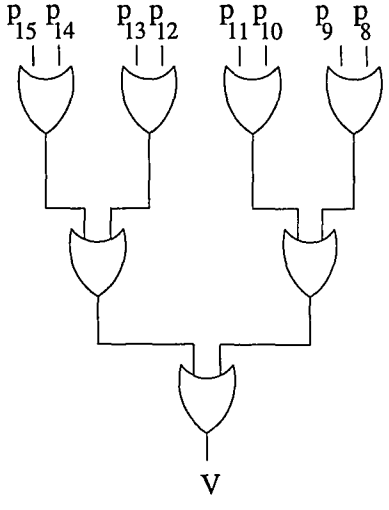

Figure 2.5: Tree of 2-input OR Gates for $n = 8$.

2.2 Two's Complement Multipliers

Two's complement numbers A and B and their product P have the values

$$\begin{aligned} A &= -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i \\ B &= -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i \\ P &= -p_{2n-1} \cdot 2^{2n-1} + \sum_{i=0}^{2n-2} p_i 2^i \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} P = {} & a_{n-1}b_{n-1}2^{2n-1} - a_{n-1}\sum_{i=0}^{n-2} b_i 2^{n+i-1} \\ & -b_{n-1}\sum_{i=0}^{n-2} a_i 2^{n+i-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} \end{aligned} \tag{2.5}$$

overflow occurs when

$$P \leq -2^{n-1} - 1 \text{ or } P \geq 2^{n-1} \tag{2.6}$$

Multiplication of two's complement numbers generates signed partial products as shown in Figure 2.6. Since $a_{n-1}b_i$ and $b_{n-1}a_i$ (for $0 \leq i \leq n-2$) have negative weights, they should be subtracted, rather than added. This makes the design difficult to implement because it requires adder and subtracter cells. Consequently, several techniques have been proposed to handle partial products with negative and positive weight, such as the Baugh-Wooley Algorithm[6] and its variations [7], [8] and, Booth's Algorithm [16]. The Baugh-Wooley algorithm provides a method

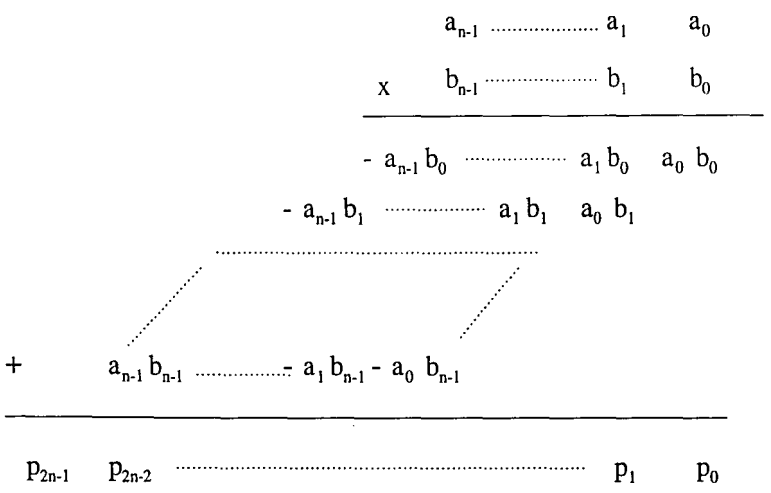

Figure 2.6: Two's Complement Multiplication Matrix.

for modifying the partial product matrix, so that all the partial product bits have positive weights. This algorithm and its modified form are often used to perform two's complement multiplication.

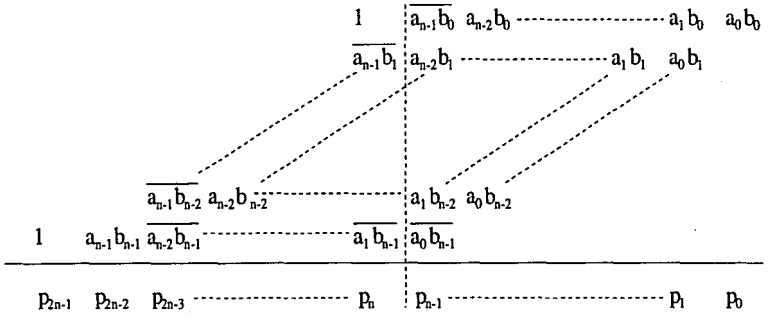

Figure 2.7: Modified Two's Complement Multiplication Matrix.

Two's complement multiplication is often realized using a variation of the Baugh-Wooley algorithm called the Complemented Partial Product Word Correction Algorithm. With this implementation, partial product bits containing a_{n-1} or b_{n-1}, but not both, are complemented and ones are added to columns n and $2n-1$. This is equivalent to taking the two's complement of the two negative terms in Equation

2.5. The multiplication matrix for this implementation is shown in Figure 2.7.

2.2.1 Two's Complement Array Multipliers

The design of an array multiplier that uses the Complemented Partial Product Word Correction Algorithm and conventional overflow detection is shown in Figure 2.8. The design shown in this figure is similar to the unsigned array multiplier design in Figure 2.2. AND gates in the left-most column are replaced by NAND gates and the last row of MFAs are replaced by Negating Modified Full Adders (NMFA). The specialized half adder (SHA) in the bottom right corner is a half adder that takes the sum and carry bits of the previous row and adds them with '1'. This cell has approximately the same area and delay as a regular half adder. The last product bit p_{2n-1} is inverted to add the one in column $2n-1$. Inverting p_{2n-1} has the same effect as adding one in column $2n-1$, because the carry out from this column is ignored.

In Figure 2.8, the bottom two rows of cells, consisting of n XOR gates and $n-1$ OR gates, are dedicated to overflow detection. The XOR gates identify whether p_{n-1} differs from any of the more significant product bits p_n to p_{2n-1}. The outputs from the XOR gates are combined to determine if the overflow flag V should be set. The logic equation for the overflow detection flag is

$$V = \widehat{p}_{2n-1} + \widehat{p}_{2n-2} \cdots + \widehat{p}_{n+1} + \widehat{p}_n \tag{2.7}$$

where $\widehat{p}_i = p_i \oplus p_{n-1}$ for $i = n$ to $2n-1$, and $\oplus$ indicates exclusive-or.

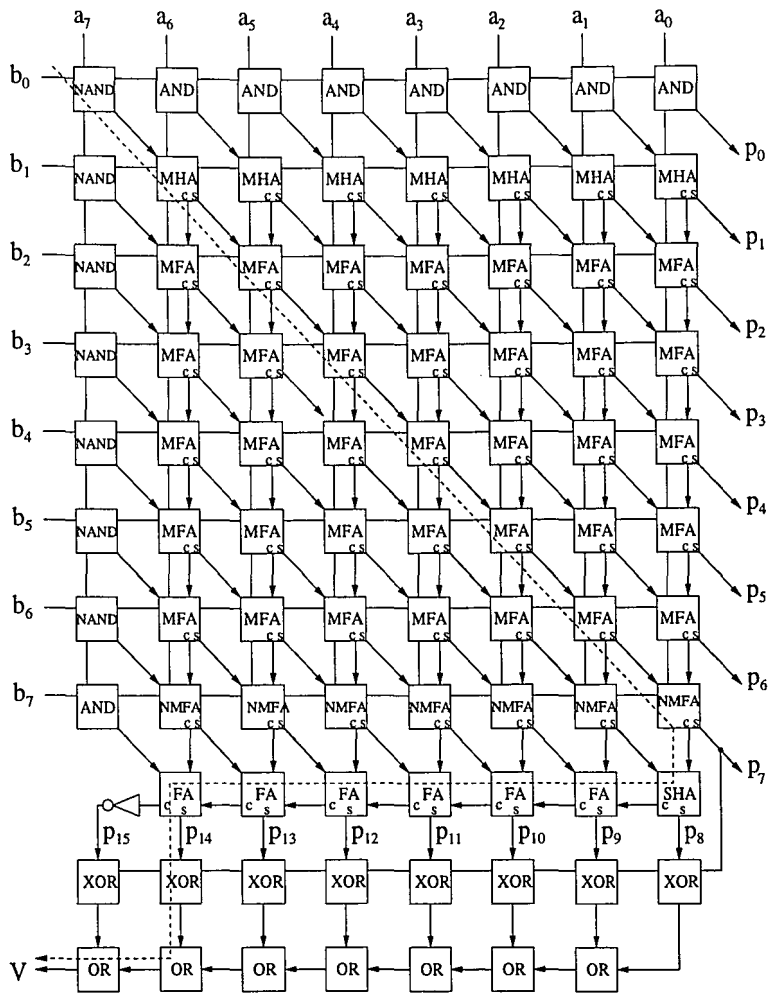

Figure 2.8: Two's Complement Array Multiplication with Conventional Overflow Detection.

Saturating multiplication is implemented by adding an n-bit 2-to-1 multiplexer, as shown in Figure 2.9. For two's complement multiplication, if the product overflows, the saturated product is determined from the sign bits of A and B. If $V = 1$ and $a_{n-1} \oplus b_{n-1} = 1$, then negative overflow has occurred and the product saturates to -2^{n-1}. On the other hand, if $V = 1$ and $a_{n-1} \oplus b_{n-1} = 0$, positive overflow has occurred and the product saturates to $2^{n-1} - 1$. If $V = 0$, then overflow has not

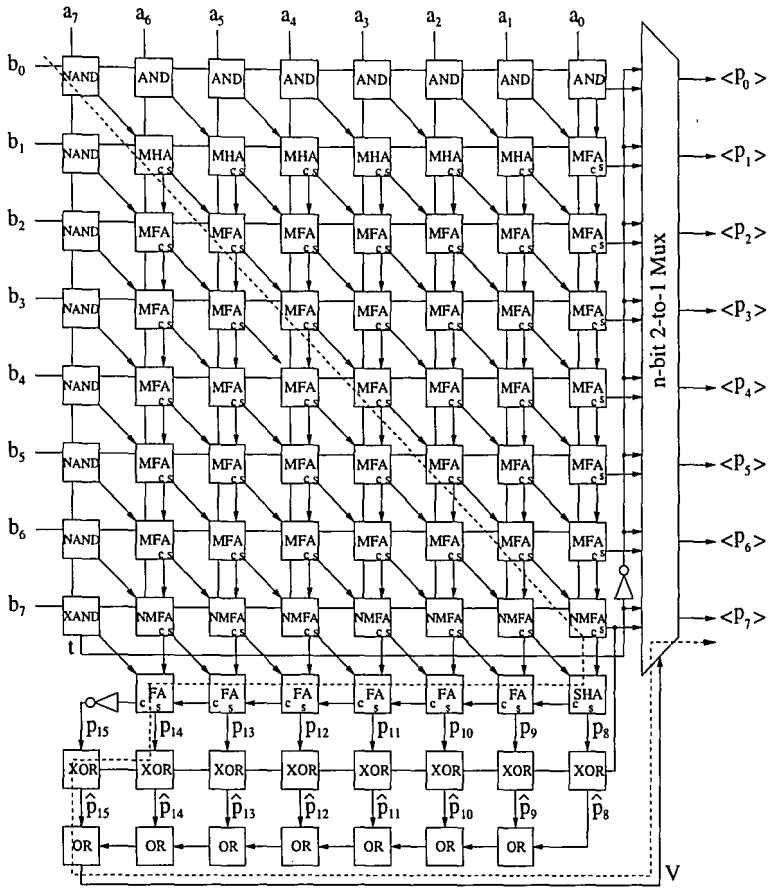

Figure 2.9: Conventional Two's Complement Saturation.

occurred and the saturated product is the n least significant bits. If $t = a_{n-1} \oplus b_{n-1}$ the saturated product $\langle p \rangle = \langle p_{n-1} \rangle \langle p_{n-2} \rangle \ldots \langle p_0 \rangle$ is

$$P = \begin{cases} p_{n-1}p_{n-2}\ldots p_0 & (\text{when } V = 0) \\ t\bar{t}\ldots\ldots\bar{t} & (\text{when } V = 1) \end{cases} \tag{2.8}$$

2.2.2 Two's Complement Tree Multipliers

Several techniques are available for implementing two's complement multiplier trees [4], [8]. Figure 2.7 shows the dot diagram of an 8 by 8 two's complement multiplier tree that uses the Complemented Partial Product Algorithm [8] and Dadda's reduction method [1]. Similar to the two's complement array multiplier, $(2n-2)$ of the partial product bits are inverted and ones are added to columns n and $(2n-1)$. Although it seems that the heights of these two columns are increased by adding ones, this does not effect adversely the design, because the most significant product bit is simply inverted and a SHA adds one in the column n. In Figure 2.10, a dot with a line above it indicates a complemented partial product bit and the circled half adder in column 8 is a (SHA) specialized half adder. An n by n two's complement Dadda tree multiplier has $(2n-1)$ inverters, n^2 AND gates , $(n-1)$ HAs, (n^2-4n+3) FAs, and a $(2n-2)$ bit CPA.

Conventional techniques for overflow detection and saturation for two's complement tree multipliers are similar to the techniques used for two's complement array multipliers. The only difference is that tree multipliers tend to use a tree of OR gates, rather than a linear array of OR gates, when computing the overflow flag.

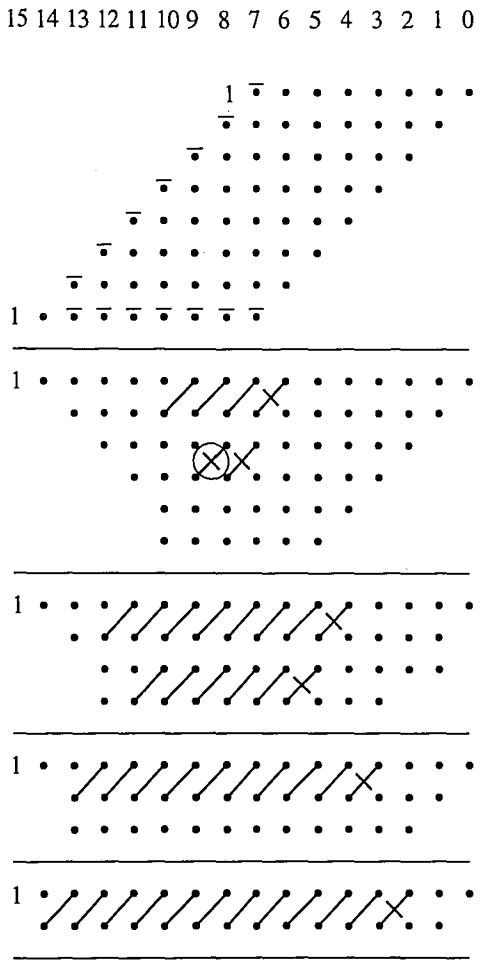

Figure 2.10: Two's Complement Dadda Tree Multiplier.

Chapter 3

Overflow Detection and Saturation for Unsigned Integer Multiplication

3.1 General Design Approach

Instead of computing all $2n$ bits of the product, the methods proposed in this thesis only compute the n least significant product bits and have separate overflow detection logic, as shown in Figure 3.1. Carries into column n are also used in the overflow detection circuit.

The main idea behind the proposed unsigned overflow detection methods is that overflow occurs if any of the partial product bits in column n to $(2n-2)$ are '1' or any

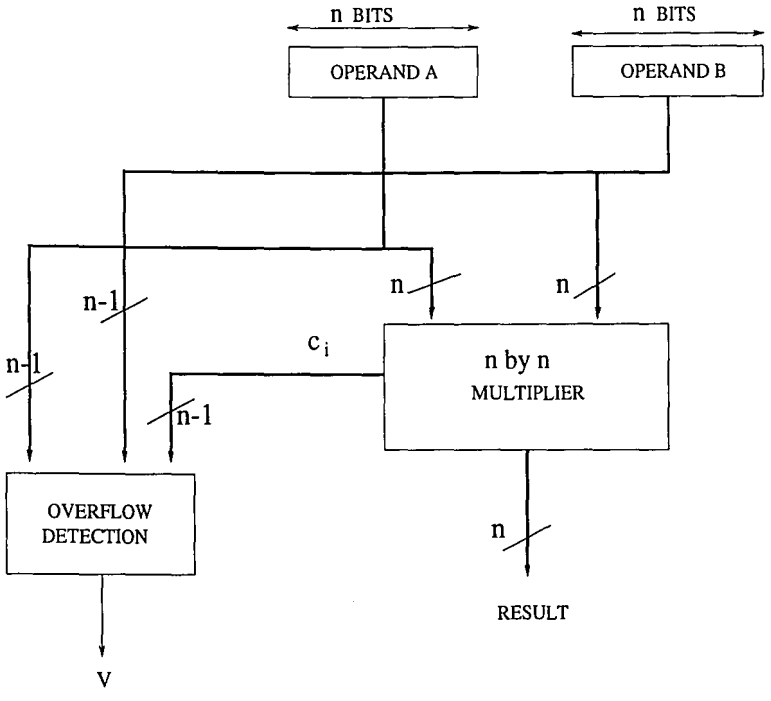

Figure 3.1: Block Diagram for Unsigned Multiplication Overflow Detection.

of the carries into column n are '1'. Consequently, these ones can be detected without adding the partial products. The logic equation for unsigned overflow detection is

$$V = \sum_{i=1}^{n-1}(c_{i+1} + \sum_{j=1}^{i} a_{n-j} \cdot b_i) \tag{3.1}$$

In this expression, V is the overflow flag, c_i is the i^{th} carry into column n, bit summations corresponds to logical ORs and bit multiplications corresponds to logical ANDs.

3.2 Unsigned Array Multipliers with Overflow Detection or Saturation

Figure 3.2. shows an 8 by 8 multiplication matrix to demonstrate how the partial product bits are used to detect overflow with the proposed method.

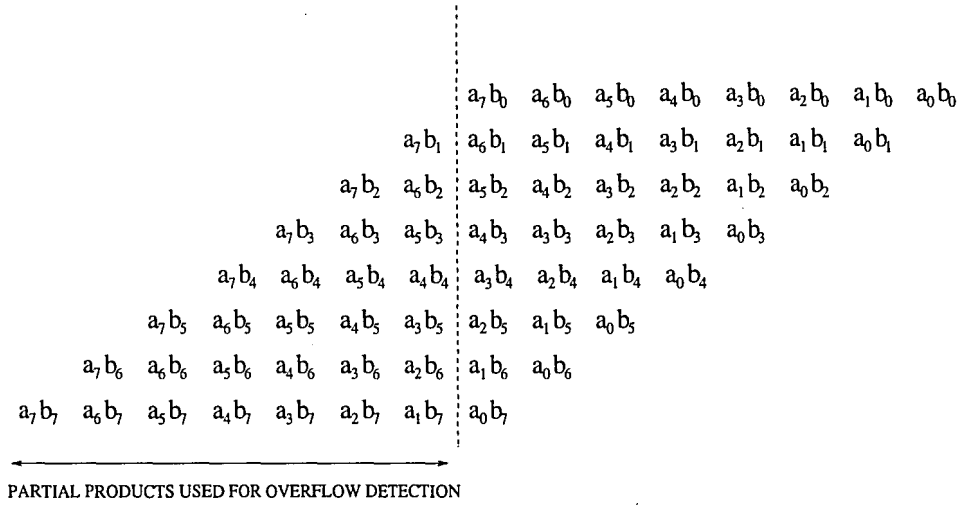

Figure 3.2: 8 by 8 Unsigned Multiplication Matrix.

Using Equation 3.1, with $n = 8$, gives

$$\begin{aligned} V &= b_1 \cdot a_7 + c_2 \\ &+ b_2 \cdot (a_7 + a_6) + c_3 \\ &+ b_3 \cdot (a_7 + a_6 + a_5) + c_4 \\ &+ b_4 \cdot (a_7 + a_6 + a_5 + a_4) + c_5 \\ &+ b_5 \cdot (a_7 + a_6 + a_5 + a_4 + a_3) + c_6 \\ &+ b_6 \cdot (a_7 + a_6 + a_5 + a_4 + a_3 + a_2) + c_7 \\ &+ b_7 \cdot (a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1) + c_8 \end{aligned} \tag{3.2}$$

Common terms in the logic equation for overflow detection are used to reduce the hardware needed to detect overflow. An overflow detection circuit constructed using AND and OR gates is shown in the Figure 3.3 for an 8 by 8 unsigned multiplication. For an n by n multiplier, the three gates in the dashed lines are replicated $n-2$ times. These three gates are combined to form an overflow detection (OVD) cell. Overflow is detected using the following iterative equations.

$$\begin{aligned} o_{i+1} &= o_i + a_{n-i} \\ v_{i+1} &= v_i + c_i + o_{i+1} \cdot b_i \end{aligned} \tag{3.3}$$

for $2 \leq i \leq n-1$, where o_i is a temporary OR bit with an initial value of $o_2 = a_{n-1}$, and v_i is a temporary overflow bit with an initial value $v_2 = a_{n-1} \cdot b_1$. The o_i and v_i bits are shown in Figure 3.3.

An OVD cell takes a_{n-i}, b_i, c_i and v_i as inputs and generates o_{i+1} and v_{i+1} as outputs. Each OVD cell contains one AND gate, one 2-input OR gate, and one 3-input OR gate. To form an unsigned multiplier with the proposed overflow detection method, these cells are combined with an unsigned array multiplier from which the cells used to compute p_n to p_{2n-1} have been removed. This is shown in Figure 3.4 for an 8 by 8 unsigned array multiplier.

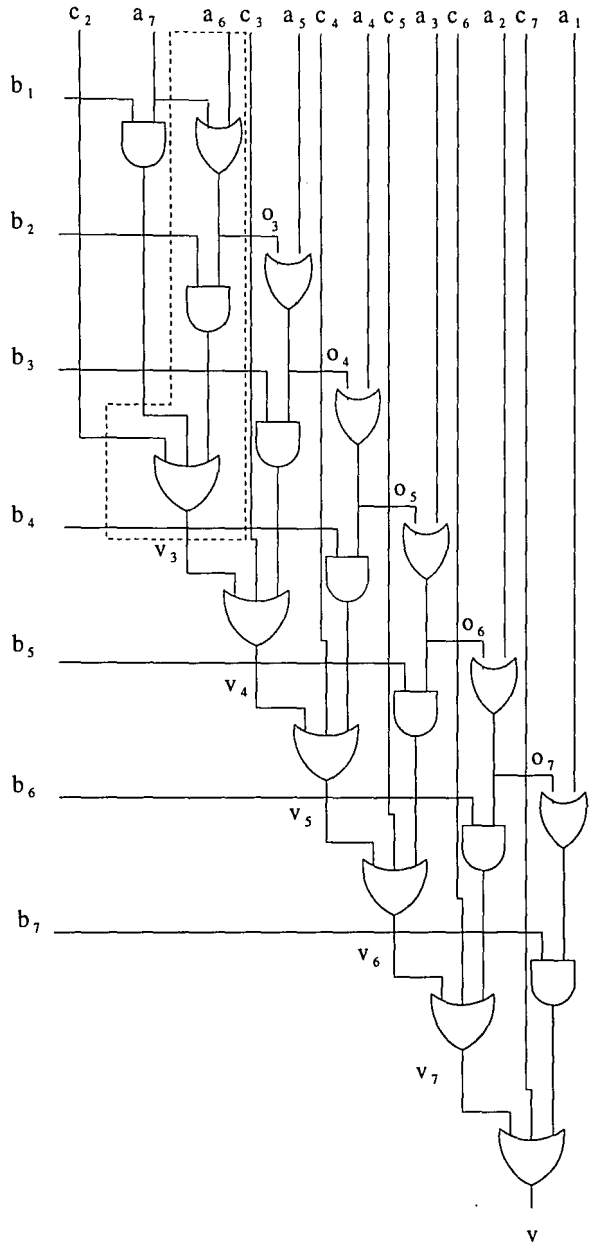

Figure 3.3: Proposed Overflow Detection Logic for $n = 8$.

An n-bit unsigned array multiplier that uses the proposed method for overflow detection requires $(n^2 + 3n - 2)/2$ AND gates, $(n - 1)$ 2-input OR gates $(n - 1)$ HAs and $(n^2 - 3n + 2)/2$ FAs. This corresponds to $(n^2 - n)/2$ fewer adders, $(n^2 - 3n + 2)/2$ fewer AND gates, and $(n - 2)$ more 3-input OR gates than the conventional method.

The worst case delay path is indicated by the dashed line in Figure 3.4. Since a MFA has longer delay than an OVD cell, the unsigned multiplier with the proposed overflow detection logic has a delay approximately half as long as the unsigned multiplier with conventional overflow detection, shown in Figure 3.3.

Unsigned saturating multiplication using the proposed method is performed by ORing the overflow bit with n least significant product bits, as shown in Figure 3.5. If the overflow bit is '1' this produces a product with n ones, which corresponds to the maximum representable unsigned number; otherwise the product is not changed. This requires n more OR gates and the worst case delay increases by one OR gate delay.

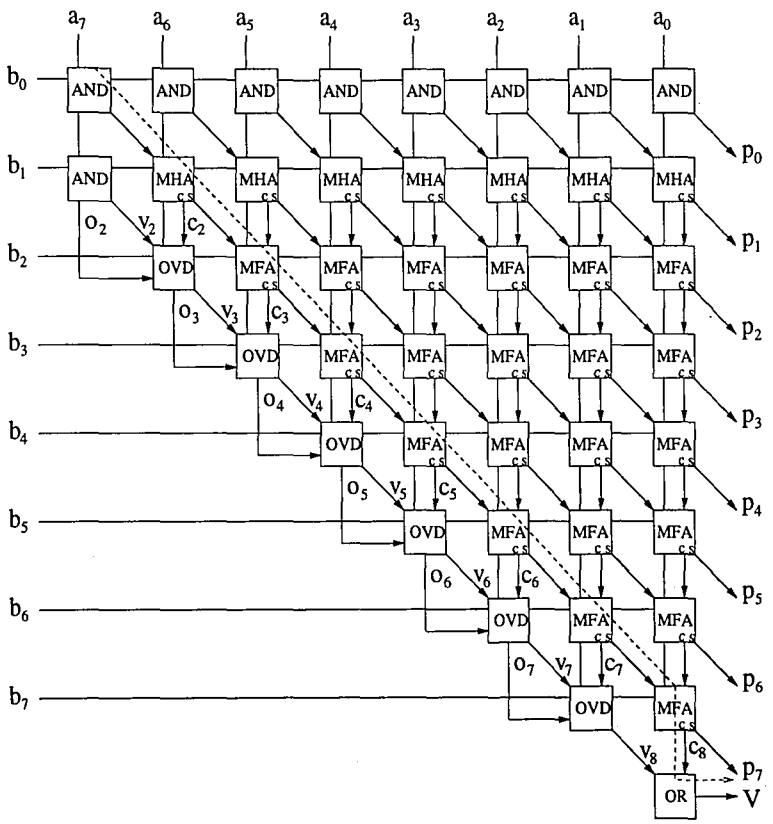

Figure 3.4: Unsigned Array Multipliers with Proposed Overflow Detection Logic.

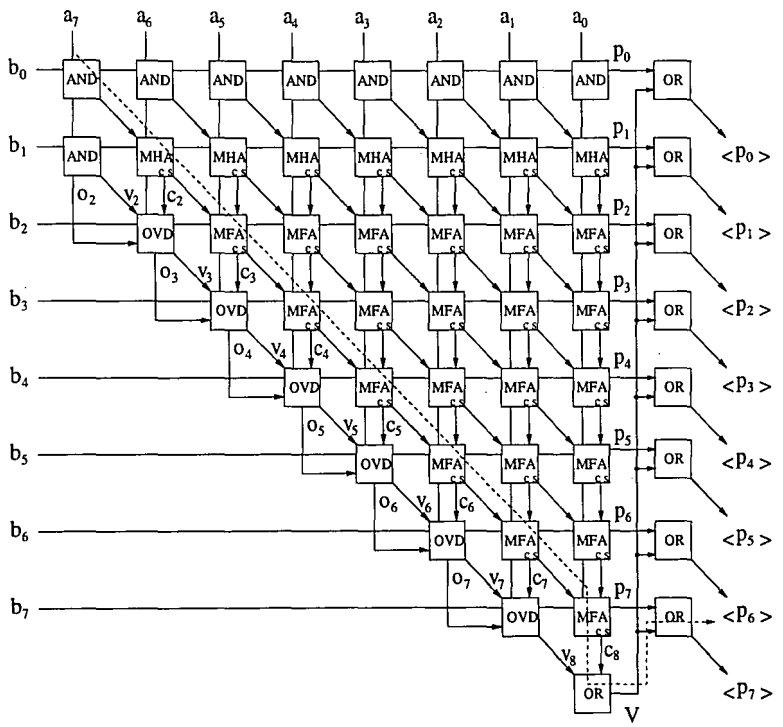

Figure 3.5: Unsigned Array Multipliers with Proposed Saturation Logic.

3.3 Unsigned Tree Multipliers with Overflow Detection or Saturation.

For unsigned tree multipliers, the technique of using an array of OVD cells with linear delay does not work well, since the OVD cells would significantly increase the multipliers' worst case delay path. Instead, all n^2 partial product bits are generated and a tree of OR gates is used to determine if any of the partial product bits in columns n to $2n-1$ or any carries into column n are one. This method is shown for an 8 by 8 multiplier in Figure 3.6, where the symbol 'o' denotes the output of a 2-input OR gate. Although this method requires more hardware for overflow detection than the unsigned array multiplier, the overflow detection logic has logarithmic delay and no longer contributes significantly to the critical path. An n by n unsigned array multiplier that uses this method has $(n^2+n-4)/2$ OR gates, n^2 AND gates, $(n-2)$ HAs, and $(n^2-5n+6)/2$ FAs. Since the delay of the OR gates for overflow detection is less than the delay of the partial product reduction stages, the worst case delay is equal to the delay of partial product generation, plus partial product reduction, plus an $(n-1)$-bit carry-propagate addition plus one OR gate delay to include the final carry out. Saturating multiplication is performed with the same method that is used by the array multiplier. The overflow bit is OR'ed with the n least significant bits of the product.

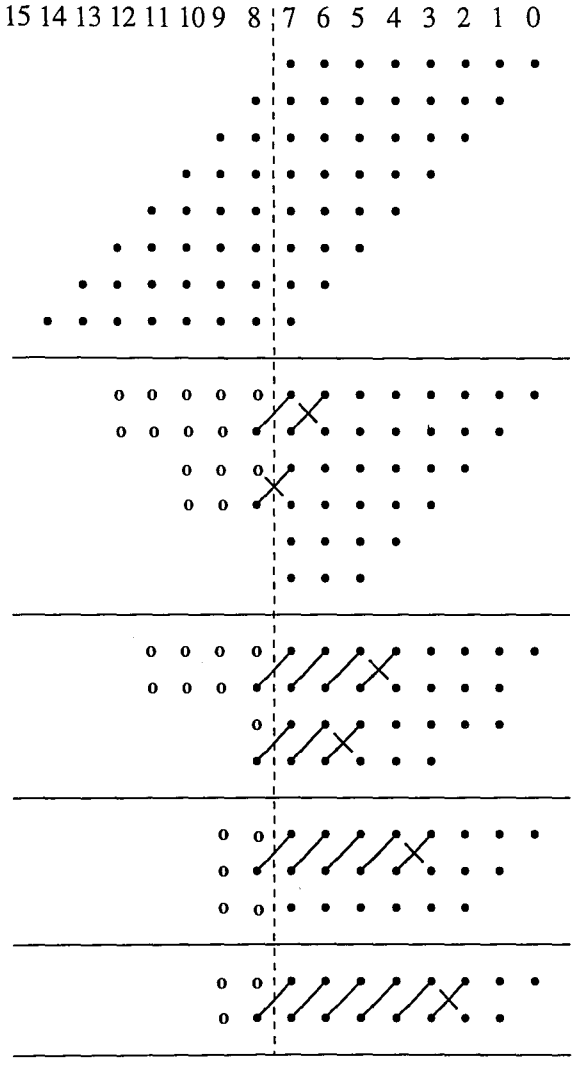

Figure 3.6: Unsigned Tree Multiplier with Proposed Overflow Detection Logic.

Chapter 4

Overflow and Saturation Detection for Two's Complement Integer Multiplication

4.1 General Approach

The proposed method for overflow detection in two's complement multiplication detects the number of consecutive bits that are equal to the sign bit. Essentially, this method counts the number of leading zeros if the operand is positive and the number of leading ones if the operand is negative. For example, 11100101 has three leading ones and 00001001 has four leading zeros. This method works because the number of leading zeros or ones indicates the magnitude of the operand; operands

with more leading zeros or ones have smaller magnitudes and therefore are less likely to cause overflow. The main issue is to determine how many of leading zeros and leading ones are needed to guarantee that overflow will occur or that overflow will not occur. A block diagram that shows the proposed approach is shown in Figure 4.1.

The analysis for two's complement multiplication has three cases depending on the operands' signs; both operands positive, both operands negative, or one positive and the other negative. Overflow regions for these three cases are discussed in the following sections.

4.1.1 Case 1: Both Operands are Positive

Let Z_A denote the number of leading zeros of operand A and Z_B denote the number of leading zeros of operand B. Since both operands are positive n-bit integers, they have at least one and at most n leading zeros, which can be expressed as

$$\begin{aligned} 1 \leq \quad & Z_A & \leq n \\ 1 \leq \quad & Z_B & \leq n \\ 2 \leq \quad & Z_A + Z_B & \leq 2n \end{aligned} \tag{4.1}$$

The ranges for operand A and operand B in terms of the number of the leading zeros are expressed as

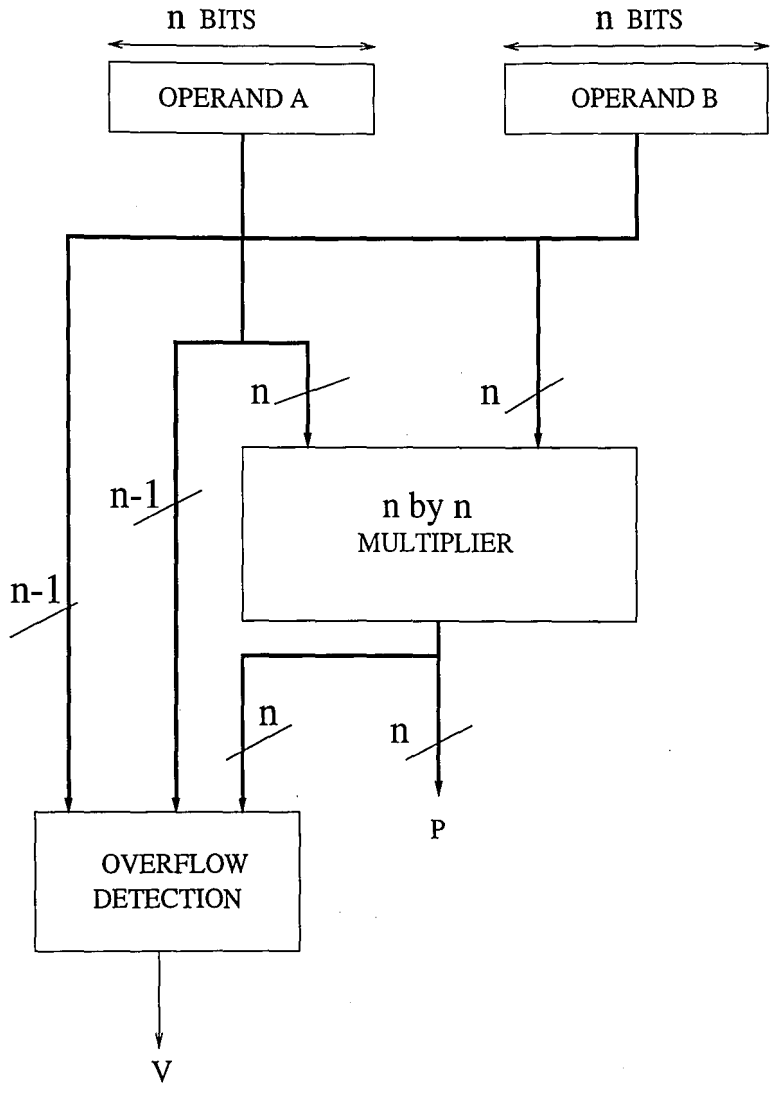

Figure 4.1: Block Diagram of The Proposed Method for Two's Complement Multiplication with Overflow Detection.

$$\begin{aligned} 2^{n-Z_A-1} \leq \quad A \quad &\leq 2^{n-Z_A} - 1 \\ 2^{n-Z_B-1} \leq \quad B \quad &\leq 2^{n-Z_B} - 1 \end{aligned} \tag{4.2}$$

Overflow occurs for Case 1 when $P = A \cdot B \geq 2^{n-1}$, and overflow does not occur when $P < 2^{n-1}$. Based on (4.2) the range of P is

$$2^{n-Z_A-1} \cdot 2^{n-Z_B-1} \leq (2^{n-Z_A} - 1) \cdot (2^{n-Z_B} - 1) \tag{4.3}$$

Using (4.3), overflow is guaranteed to occur when

$$2^{n-1} \leq \quad P \quad \leq 2^{n-Z_A-1} \cdot 2^{n-Z_B-1} \tag{4.4}$$

To determine the number of leading zeros in A and B that guarantee overflow, (4.4) is rewritten as

$$2^{n-1} \leq 2^{2n-Z_A-Z_B-2} \tag{4.5}$$

Taking the base 2 logarithm of both sides gives

$$n - 1 \leq 2n - Z_A - Z_B - 2 \tag{4.6}$$

or equivalently,

$$Z_A + Z_B \leq n - 1 \tag{4.7}$$

Thus, if A and B together have less than n leading zeros, overflow must occur. Similarly, overflow is guaranteed not to occur when

$$(2^{n-Z_A} - 1) \cdot (2^{n-Z_B} - 1) < 2^{n-1} \tag{4.8}$$

To determine the number of leading zeros in A and B that guarantee overflow does not occur, Equation 4.8 is rewritten as

$$2^{2n-Z_A-Z_B} - 2^{n-Z_A} - 2^{n-Z_B} + 1 < 2^{n-1} \tag{4.9}$$

since $-2^{n-Z_A} \leq -1$ and $-2^{n-Z_B} \leq -1$,

$$2^{2n-Z_A-Z_B} - 2^{n-Z_A} - 2^{n-Z_B} + 1 < 2^{2n-Z_A-Z_B} \tag{4.10}$$

Consequently, overflow is guaranteed not to occur if

$$2^{2n-Z_A-Z_B} < 2^{n-1} \tag{4.11}$$

Taking the base 2 logarithm of both sides gives

$$2n - Z_A - Z_B < n - 1 \tag{4.12}$$

or equivalently

$$n + 1 < Z_A + Z_B \tag{4.13}$$

Using $n + 1 = Z_A + Z_B$ in (4.9) gives

$$2^{n-1} - 2^{Z_B-1} - 2^{Z_A-1} + 1 < 2^{n-1} \tag{4.14}$$

which is always true since $Z_A \geq 1$ and $Z_B \geq 1$. Therefore, 4.13 can be rewritten as

$$n + 1 \leq Z_A + Z_B \tag{4.15}$$

Thus, overflow is guaranteed not to occur if A and B together have more than n leading zeros.

When $Z_A + Z_B = n$, it cannot be directly determined whether or not overflow has occurred only by examining the number of leading zeros. Rewriting (4.3) for $n = Z_A + Z_B$ gives

$$2^{n-2} \leq P \leq 2^n - 2^{Z_A} - 2^{Z_B} + 1 \tag{4.16}$$

This problem however can be solved by further analyzing what happens when $Z_A + Z_B = n$. The results obtained so far are summarized in Figure 4.2.

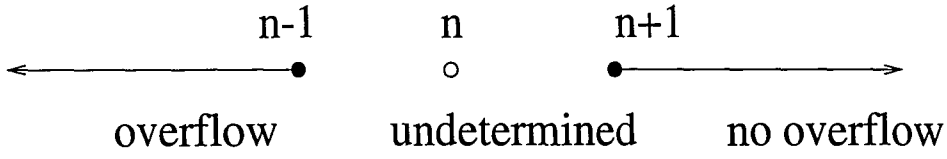

Figure 4.2: Overflow Regions for $Z_A + Z_B$.

To Determine whether or not overflow occurs when $Z_A + Z_B = n$,it is necessary to calculate how many product bits are needed to represent the result when $Z_A + Z_B = n$. Then by using the most significant product bits and the sign bits of the operands the overflow flag is set. If n is even, the maximum product is

$$P = (2^{n/2} - 1) \cdot (2^{n/2} - 1) \tag{4.17}$$

or equivalently

$$P = 2^n - 2^{(n+2)/2} + 1 \tag{4.18}$$

If n is odd the maximum product is

$$P = (2^{(n+1)/2)} - 1) \cdot (2^{(n-1)/2} - 1) \tag{4.19}$$

or equivalently

$$P = 2^n - 2^{(n+1)/2} - 2^{(n-1)/2} + 1 \tag{4.20}$$

Thus, when $Z_A + Z_B = n$, P can always be represented with n bits and overflow can be determined simply by examining the sign bit p_{n-1}. If $p_{n-1} = 1$, overflow occurs. Otherwise, overflow does not occur.

4.1.2 Case 2: Both Operands are Negative

Let I_A denote the number of leading ones of operand A and I_B denote the number leading zeros of operand B. Since both operands are negative n-bit integers, they have at least one and at most n leading ones. This can be expressed as

$$\begin{aligned} 1 \leq \quad & I_A \quad \leq n \\ 1 \leq \quad & I_B \quad \leq n \\ 2 \leq \quad & I_A + I_B \quad \leq 2n \end{aligned} \tag{4.21}$$

The ranges for A and B in terms of leading ones are expressed as

$$\begin{aligned} -2^{n-I_A} \leq \ A \ &\leq -(1+2^{n-I_A-1}) \\ -2^{n-I_B} \leq \ B \ &\leq -(1+2^{n-I_B-1}) \end{aligned} \tag{4.22}$$

Overflow occurs for Case 2 when $P = A \cdot B \geq 2^{n-1}$ and overflow does not occur when $P < 2^{n-1}$. Based on (4.22) the range of P is

$$(1+2^{n-I_A-1}) \cdot (1+2^{n-I_B-1}) \leq P \leq 2^{2n-I_A-I_B} \tag{4.23}$$

Using (4.23), overflow is guaranteed to occur when

$$2^{n-1} \leq (1+2^{n-I_A-1}) \cdot (1+2^{n-I_B-1}) \tag{4.24}$$

To determine the number of leading ones in A and B that guarantee overflow occurs (4.24) is rewritten as

$$2^{n-1} \leq 1 + 2^{n-I_A-1} + 2^{n-I_B-1} + 2^{2n-I_A-I_B-2} \tag{4.25}$$

since

$$2^{2n-I_A-I_B-2} < 1 + 2^{n-I_A-1} + 2^{n-I_B-1} + 2^{2n-I_A-I_B-2} \tag{4.26}$$

values that satisfy

$$2^{n-1} \leq 2^{2n-I_A-I_B-2} \tag{4.27}$$

also satisfy (4.25), Taking the base 2 logarithm of both sides of (4.27) gives

$$I_A + I_B \leq n-1 \tag{4.28}$$

Thus, for negative integers overflow occurs when operands have less than $n-1$ leading ones.

Using (4.23) overflow is guaranteed not to occur when

$$2^{2n-I_A-I_B} \leq P < 2^{n-1} \tag{4.29}$$

Taking the base 2 logarithm of both sides of Equation 4.29 gives

$$n+1 < I_A + I_B \tag{4.30}$$

Thus, overflow is guaranteed not to occur when A and B together have more than $n+1$ leading ones.

When $n \leq I_A + I_B \leq n+1$ it cannot be directly determined if overflow has occurred. This is seen by using $n = I_A + I_B$ in (4.23), which gives

$$1 + 2^{I_B-1} + 2^{I_A-1} + 2^{2n-2} \leq P \leq 2^n \tag{4.31}$$

similarly when $n+1 = I_A + I_B$

$$1 + 2^{I_B-2} + 2^{I_A-2} + 2^{2n-3} \leq P \leq 2^{n-1} \tag{4.32}$$

The results so far are shown in Figure 4.3.

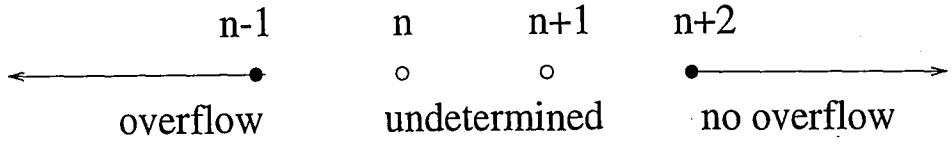

Figure 4.3: Overflow Regions for $I_A + I_B$.

When $I_A + I_B = n$ and n is even, the maximum result is

$$-2^{-(n/2)} \cdot -2^{-(n/2)} = 2^n \tag{4.33}$$

when n is odd, the maximum product is

$$-2^{(n+1)/2} \cdot -2^{(n-1)/2} = 2^n \tag{4.34}$$

Since $P \leq 2^n$ when $n \leq I_A + I_B \leq n+1$, P can be represented using only n bits, except when $P = 2^n$. Thus, for Case 2 if $n \leq I_A + I_B \leq n+1$, overflow only occurs when $p_{n-1} = 1$ or $P = 2^n$.

4.1.3 Case 3: When The Signs of The Operands Differ

Let I_A denote the number of leading ones in operand A and Z_B denote the number of leading zeros in operand B. Negative operand A has at least one and at most n leading zeros and positive operand B has at least one and at most n leading ones. This can be expressed as

$$\begin{aligned} 1 \leq \quad & I_A & \leq n \\ 1 \leq \quad & Z_B & \leq n \\ 2 \leq \quad & I_A + Z_B & \leq 2n \end{aligned} \tag{4.35}$$

The ranges for A and B in terms of leading ones and zeros are

$$\begin{aligned} -2^{n-I_A} \leq \quad & A & \leq -(1 + 2^{n-I_A-1}) \\ 2^{n-Z_B-1} \leq \quad & B & \leq 2^{n-Z_B} - 1 \end{aligned} \tag{4.36}$$

Overflow occurs for Case 3 when $P \leq -2^{n-1}-1$ and does not occur when $P \geq -2^{n-1}$. Based on (4.36) the range of P is

$$(-2^{n-I_A}) \cdot (2^{n-Z_B} - 1) \leq P \leq -(1 + 2^{n-I_A-1}) \cdot 2^{n-Z_B-1} \tag{4.37}$$

Using Equation 4.37, overflow is guaranteed to occur when

$$-(1 + 2^{n-I_A-1}) \cdot 2^{n-Z_B-1} \leq -2^{n-1} - 1 \tag{4.38}$$

or equivalently

$$2^{n-1} + 1 \leq 2^{n-Z_B-1} + 2^{2n-Z_B-I_A-2} \tag{4.39}$$

For $n-1 \leq I_A+Z_B$, (4.39) is always true, since $2^{n-1} \leq 2^{2n-Z_B-I_A-2}$ and $1 \leq 2^{n-Z_B-1}$.

Using 4.37 overflow is guaranteed not to occur when

$$-2^{n-1} \leq (-2^{n-I_A}) \cdot (2^{n-Z_B} - 1) \tag{4.40}$$

or equivalently

$$2^{2n-Z_B-I_A} - 2^{n-I_A} \leq 2^{n-1} \tag{4.41}$$

For $n + 1 \leq I_A + Z_B$, (4.41) is always true, since $2^{2n-Z_B-I_A} \leq 2^{n-1}$ and $2^{n-I_A} \geq 1$ in this range.

When $n = I_A + Z_B$ it cannot be determined whether or not overflow occurred since for $n = I_A + Z_B$ (4.37) gives

$$-2^n + 2^{n-I_A} \leq P \leq -2^{n-Z_B-1} - 2^{n-2} \tag{4.42}$$

The Figure 4.4 shows graphically the results obtained so far for Case 3.

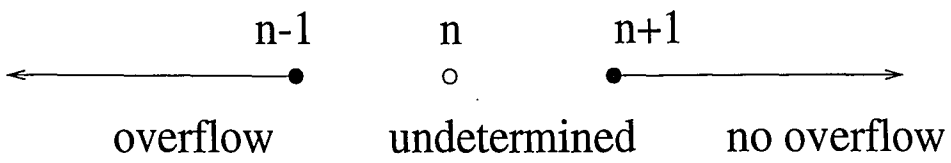

Figure 4.4: Overflow Regions for $I_A + Z_B$.

When $I_A + Z_B = n$ and n is even, the minimum negative number is

$$P = (-2^{n/2}) \cdot (2^{n/2} - 1) = -2^n + 2^{n/2} > -2^n \tag{4.43}$$

and when is n is odd it is

$$P = (-2^{(n-1)/2)}) \cdot (2^{n-(n+1)/2} - 1) = -2^n + 2^{(n-1)/2} > -2^n \tag{4.44}$$

For both cases, the product can be represented using only by n bits. Therefore, overflow occurs if $p_{n-1} = 0$.

So far, the proposed method has been explained mathematically. In the next section implementations of the overflow logic are presented.

4.1.4 Overflow Detection Logic

To allow positive and negative operands to use the same hardware for detecting leading zeros or ones, the sign bits are XNOR'ed with the $n - 2$ remaining bits.

This takes $2(n-2)$ XNOR gates and is expressed logically as

$$\widehat{a}_i = \overline{a_{n-1} \oplus a_i} \text{ for } 1 \leq i \leq n-2 \tag{4.45}$$

$$\widehat{b}_i = \overline{b_{n-1} \oplus b_i} \text{ for } 1 \leq i \leq n-2 \tag{4.46}$$

A logic design that detects $(n-1)$ or fewer leading zeros or leading ones includes $2(n-3)$ AND gates. These AND gates are used to compute

$$x(i) = \prod_{k=0}^{i} \widehat{a}_{(n-k)-2} \tag{4.47}$$

$$y(i) = \prod_{k=0}^{i} \widehat{b}_{(n-k)-2} \tag{4.48}$$

For $1 \leq i \leq n-3$. A preliminary overflow flag is generated, using $x(i)$ and $y(i)$ as

$$V_1 = \sum_{i=0}^{n-3} \overline{x(i)} \cdot \overline{y(n-i-3)} \tag{4.49}$$

where bit products correspond to logical ANDs, and bit summations correspond to logical ORs. This equation is implemented by using $(n-2)$ 2-input NOR gates and $(n-3)$ 2-input OR gates. V_1 is one when the total number of leading zeros and leading ones is less than n. In this case, overflow is guaranteed to occur. Additional logic is used to detect overflow for the undetermined regions for Cases 1-3.

For Case 1 (when $a_{n-1} = 0$ and $b_{n-1} = 0$), overflow occurs when $p_{n-1} = 1$. This is detected as

$$V_2 = (\overline{a}_{n-1} \cdot \overline{b}_{n-1} \cdot p_{n-1}) \tag{4.50}$$

For Case 2 when $a_{n-1} = 1$ and $b_{n-1} = 1$, overflow occurs when $p_{n-1} = 1$ and neither A nor B is zero, or the n least significant bits are zero.

$$V_3 = (a_{n-1} \cdot b_{n-1} \cdot p_{n-1}) \cdot \overline{Z_1} \tag{4.51}$$

$$Z_1 = \overline{a}_{n-1} \cdot x(n-3) + \overline{b}_{n-1} \cdot y(n-3) \tag{4.52}$$

$$Z_2 = \overline{p_0 + p_1 + \ldots + p_{n-1} + \overline{a_{n-1}} + \overline{b_{n-1}}} \tag{4.53}$$

For Case 3, (when $t = a_{n-1} \oplus b_{n-1} = 1$), overflow occurs when $p_{n-1} = 0$ and neither A nor B is zero. This is detected as

$$V_4 = t \cdot \overline{p}_{n-1} \cdot \overline{Z}_1 \tag{4.54}$$

Logic Equations 4.50 through 4.54 can be realized by 9 2-input AND gates, 4 inverters and $(n+1)$ OR gates. The final overflow flag V is generated by OR'ing all of the previous flags.

$$V = V_1 + V_2 + V_3 + V_4 + Z_2 \tag{4.55}$$

The overflow detection circuit requires, $(2n-4)$ 2-input XNOR gates, $(2n+3)$ 2- input AND gates, $(2n-2)$ 2-input OR gates, $(n-2)$ 2-input NOR gates, and four inverters. An overflow detection circuit for an 8-bit two's complement multiplier is shown in Figures 4.5 and 4.6.

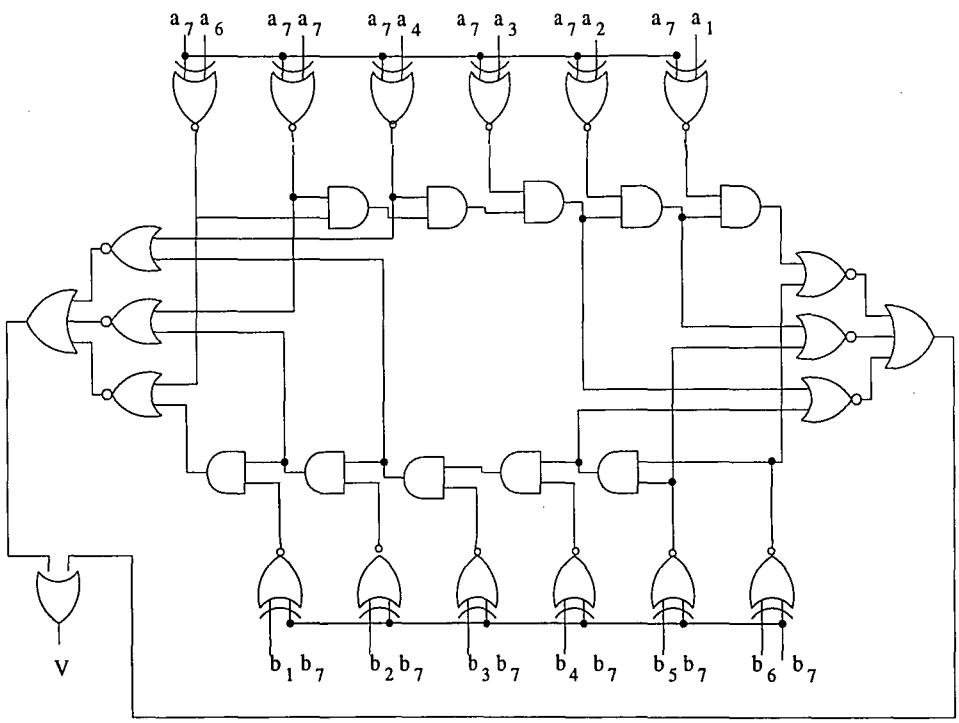

Figure 4.5: The Logic for V_1 for 8-bit Multiplication.

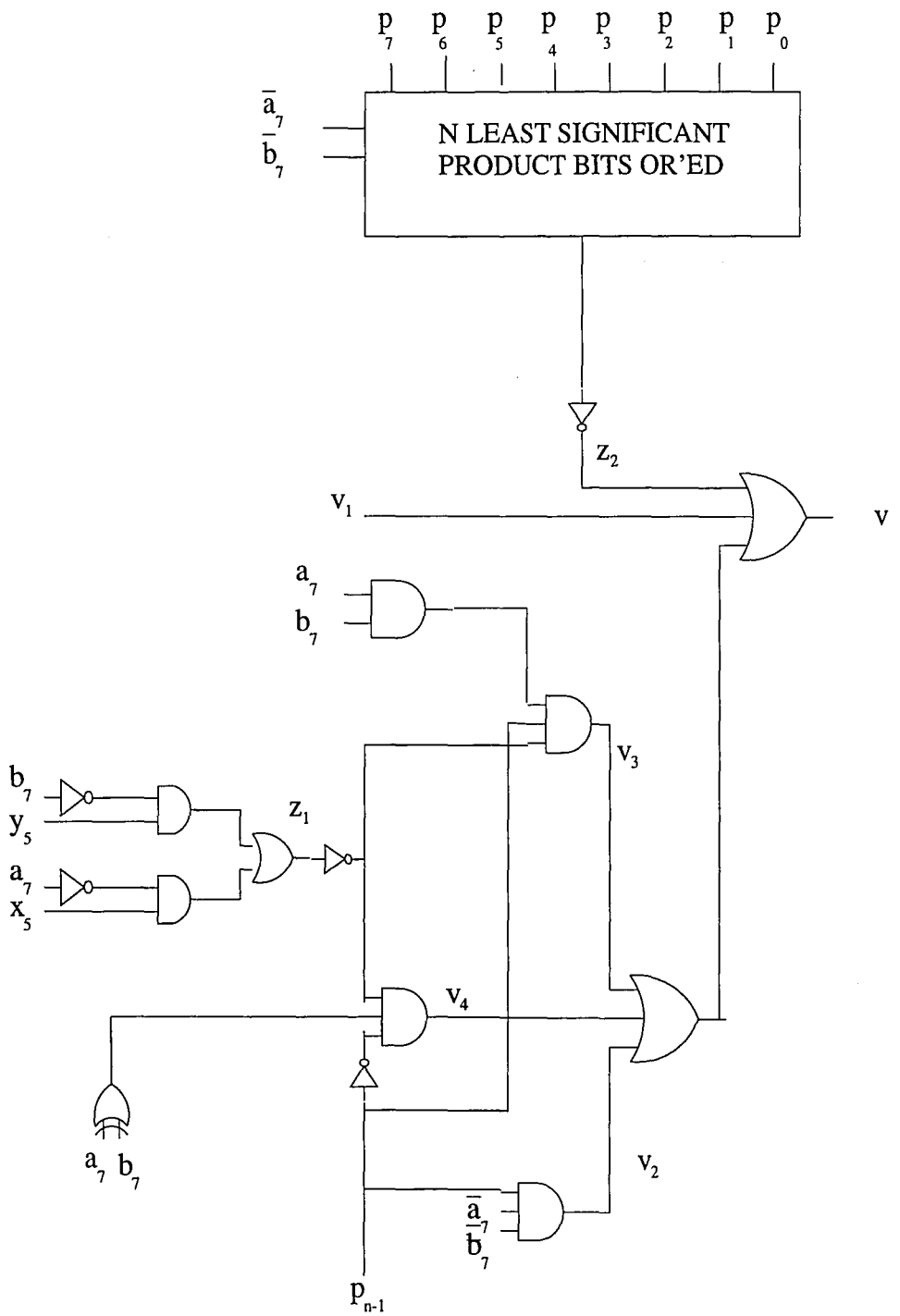

Figure 4.6: Overflow Detection Logic for 8-bit Two's Complement Multiplication.

4.1.5 An Alternative Method

An alternative method for detecting overflow in the undetermined case is, instead of generating V_2, V_3, V_4, Z_2 and n product bits, have the multiplier generate $n+1$ product bits and detect the undetermined cases by checking if $p_n \oplus p_{n-1} = 1$. This approach is shown in Figure 4.7. This approach works since for the undetermined cases we have the following situations;

(1) Case 1: $p_n = 0$ always and $p_{n-1} = 1$ only when overflow occurs.

(2) Case 2: $p_n = 0$ always and $p_{n-1} = 1$ only when overflow occurs. The one exception is when $P = 2^n$ is generated and then $p_n = 1$ and $p_{n-1} = 0$.

(3) Case 3: $p_n = 1$ always and $p_n = 0$ only when overflow occurs. For all three cases, overflow can be detected as

$$V = V_1 + (p_n \oplus p_{n-1}) \tag{4.56}$$

4.2 Two's Complement Array Multipliers with Overflow Detection or Saturation

The proposed method for overflow detection for array multipliers requires half as much hardware as the conventional method. An 8-bit two's complement multiplier

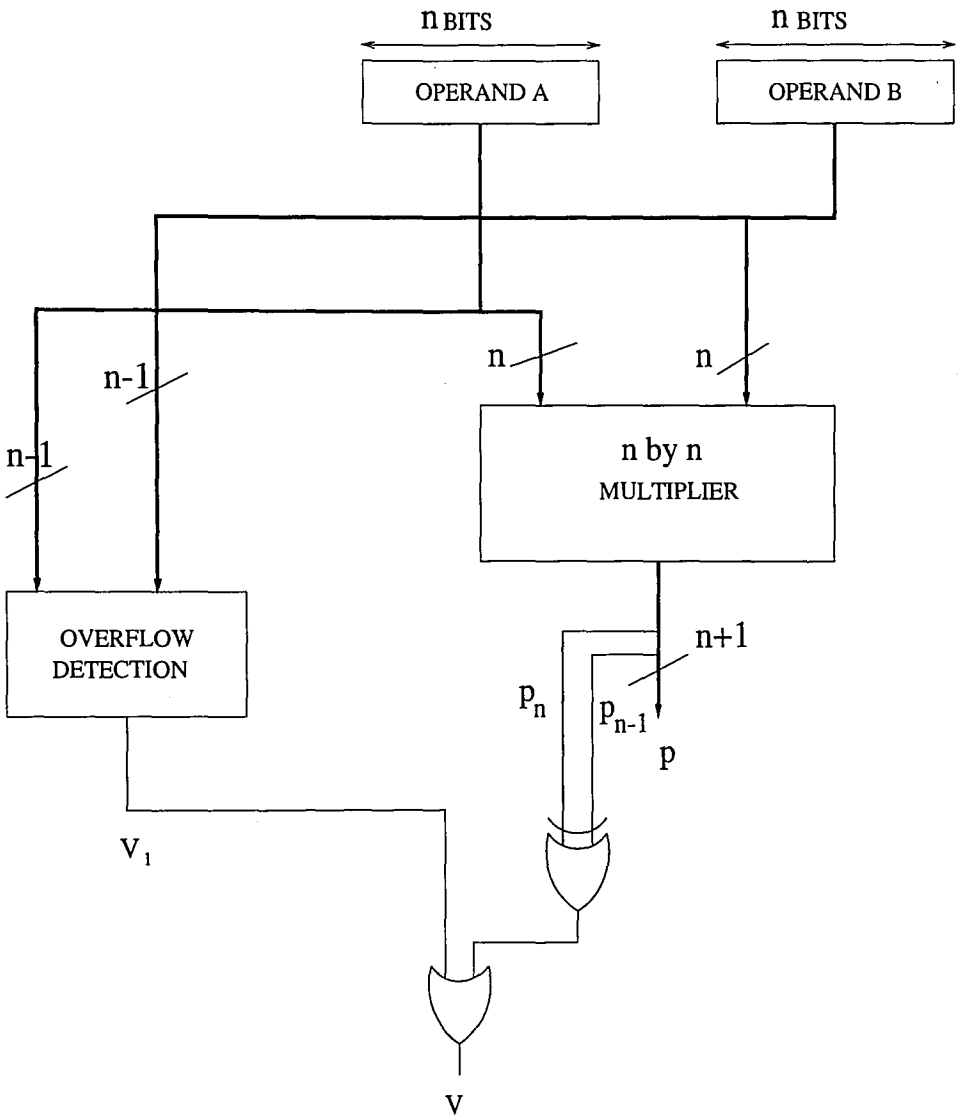

Figure 4.7: Block Diagram of the Proposed Alternative Method for Two's Complement Multiplication with Overflow Detection.

with the proposed method for overflow detection is shown in Figure 4.8. The X2A cell contains one 2-input XOR gate and one 2-input AND gate, and each X3A cell contains one 3-input XOR gate and one 2-input AND gate, an X3NA contains two 2-input XOR gate and one 2-input NAND gate.

A two's complemented array multiplier with the proposed overflow detection logic has, $(n^2-5n+6)/2$ FAs, $(n-2)$ HAs, $(2n-4)$ 2-input XNOR gates, $(n^2+5n+4)/2$ 2-input AND gates, $(2n-2)$ 2-input OR gates, $(n-2)$ 2-input NOR gates, $(n-2)$ 3-input XOR gates, one 2-input XOR gate and four inverters.

The delay for this multiplier is approximately equal to the delay of $(n-3)$ FAs plus four 2-input OR gates, plus three 3-input AND gates. The actual delay may differ according to various design decisions and the technology used.

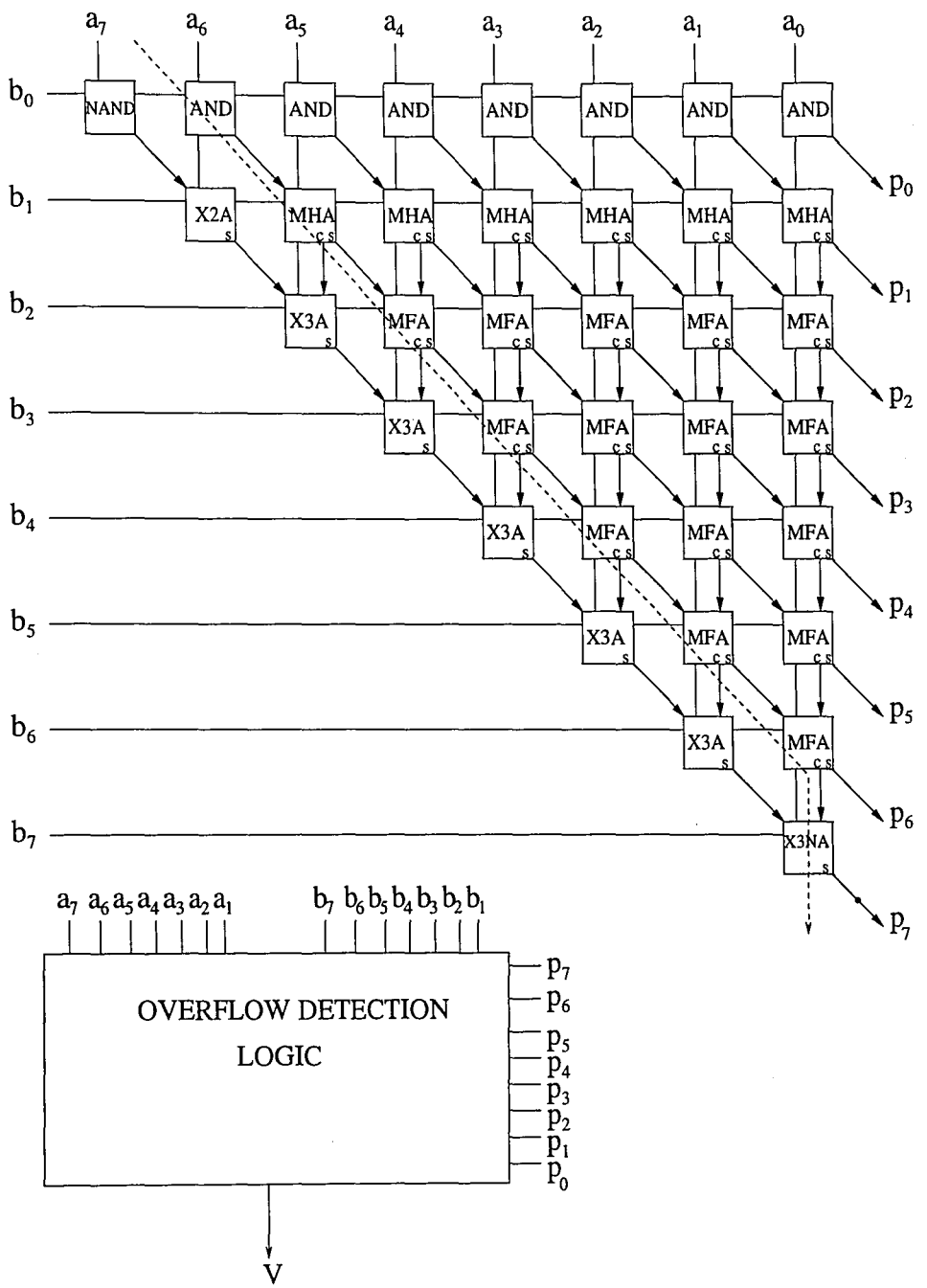

Figure 4.8: Overflow Detection Logic for 8-bit Two's Complement Multiplication.

Two's complement saturating multiplication is performed by using the V and t flags. These flags are used as inputs to an n-bit 2-to-1 multiplexer as shown in Figure 4.9. When negative overflow occurs, the result is saturated to $-2^{n-1} = 100\ldots0$ and when positive overflow occurs, result is saturated to $2^{n-1}-1 = 011\ldots1$. Adding saturation logic to the array multiplier with overflow detection only requires the addition of an inverter and an n-bit 2-to-1 multiplexor. Since V and t are already generated by the overflow detection logic, they do not require any additional hardware. The delay increases just by the delay of a 2-to-1 multiplexer plus the delay of an inverter.

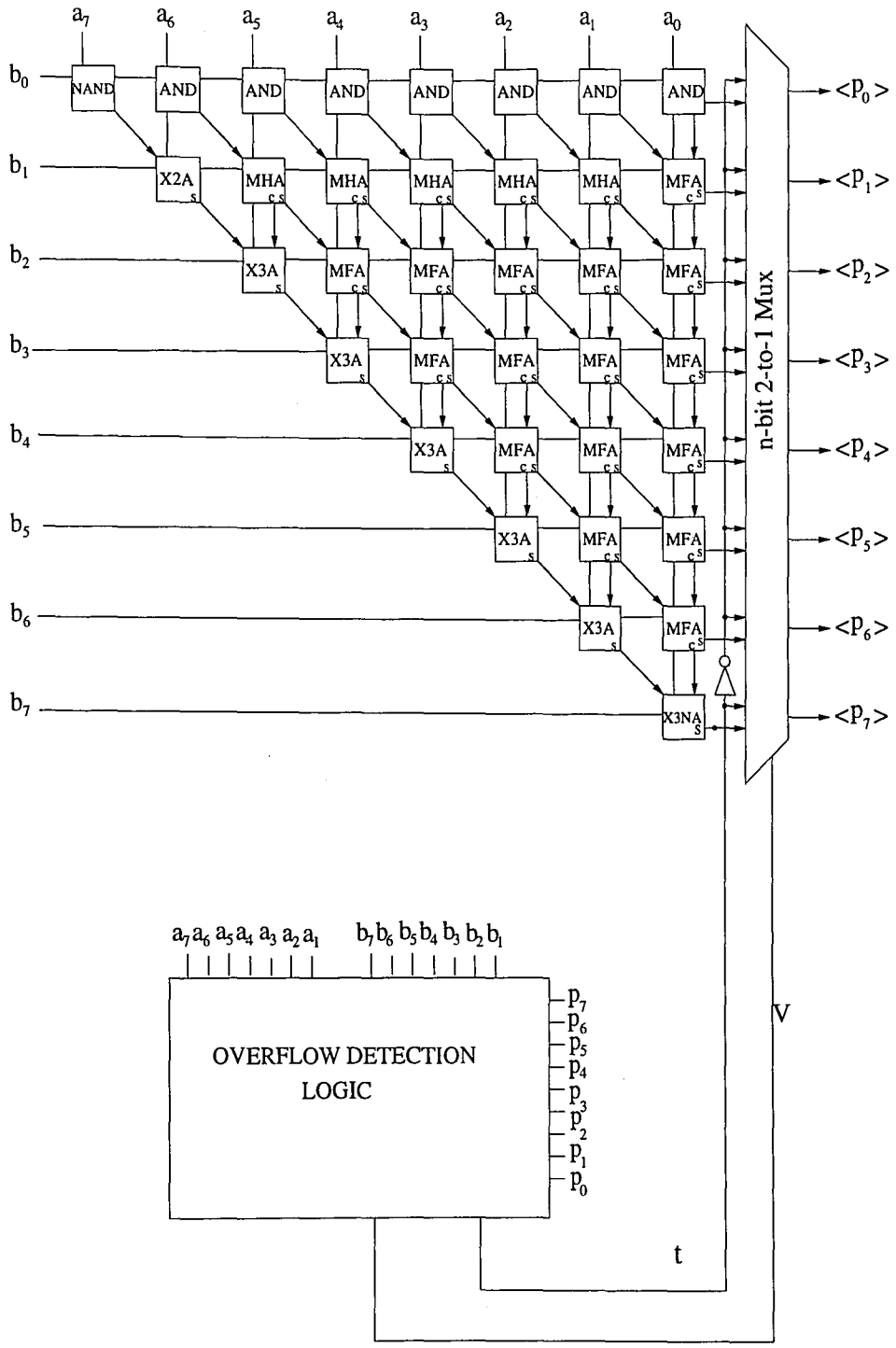

Figure 4.9: Saturating Two's Complement Array Multiplication.

4.3 Two's Complement Tree Multipliers with Overflow Detection or Saturation

The alternative method is used for tree multipliers. For this method $(n+1)$ bits of the product are computed. V_1 is computed the approach same as the first method and then the overflow flag is generated by using the logic equation $V = V_1 + (p_n \oplus p_{n-1})$, as explained in Section 4.1.5.

Since the detection circuit is independent of the multiplication process, only $n+1$ of the partial product bits needs to be generated. Consequently, the AND gates and counters that generate and reduce the partial product bits after column n of the multiplication matrix are no longer needed. The size of the carry-propagate adder is reduced to n bits since only the $n+1$ least significant product bits are used. These reductions are independent of the strategy used to design the tree multiplier. The reduction scheme of a Dadda multiplier that uses the alternative method is shown in Figure 4.10. A diagonal line with an x at the bottom is an 3 input XOR gate and a diagonal with a tilda on it and an x at the bottom represents a 2-input XNOR gate. The X's are used to denote that a carry output is not required.

If the worst case delay is the main constraint of the custom design, the alternative design method should be considered to implement the overflow detection logic. A two's complement Dadda multiplier with proposed overflow detection has $(n^2-5n+6)/2$ FAs, $(n-2)$ HAs, $(2n-4)$ 2-input XNOR gates, $(2n-5)$ 2-input XOR gates,

$(n^2+7n+4)/2$ 2-input AND gates, $(n-2)$ 2-input OR gates, $(n-2)$ 2-input NOR gates and one n-bit CPA. The worst case delay of the multiplier is also less than the conventional technique. With the alternative method,the worst case delay equals the delay of the reduction stages, plus the delay of the $n+1$-bit carry-propagate adder, plus one 2-input OR gate, plus, one 2-input XOR gate, plus one 2-input AND gate.

Two's complement saturating multiplication logic for the Dadda tree multiplier is similar to the logic for the array multiplier. An n-bit 2-to-1 multiplexer and an inverter are added as shown in Figure 4.9, except the partial product bits p_{n-2} to p_0 are not connected to the overflow detection logic. The control signals t and V for the 2-to-1 multiplexer are generated by the detection logic. The delay increases by just the delay of the inverter plus the delay of the 2-to-1 multiplexer.

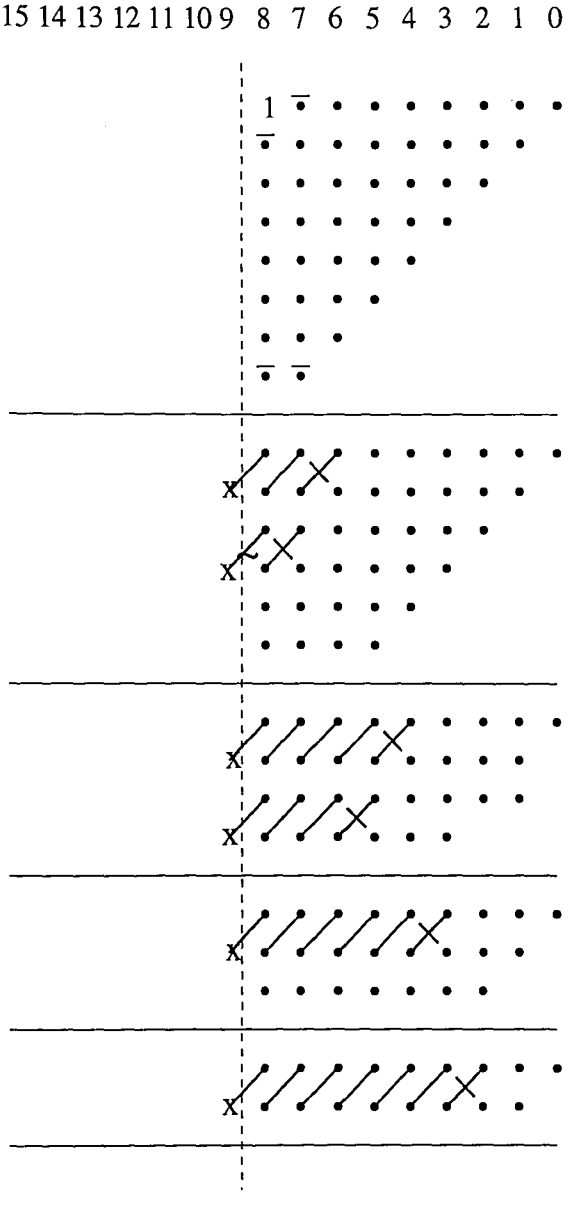

Figure 4.10: Dadda Dot Product Scheme after Proposed Overflow Detection.

Chapter 5

Results

5.1 Area and Delay Estimates

Theoretical component counts and worst case delays are given for various multipliers in Tables 5.1, 5.2 and table 5.3. In these tables, U and S denotes unsigned and signed, A and T denote array and tree multiplier, and P and C denote proposed and conventional. Table 5.1 and Table 5.2 gives the number of each component and the size of the CPA based on operand length n. Table 5.3 gives the number of each type of component on the worst case delay path. The proposed methods reduce the number of AND gates and FAs for array multipliers and reduce the number of AND gates and FAs, and the size of the CPAs for tree multipliers. The proposed methods also reduces the delays of the array and tree multiplier, since the most significant product bits are no longer calculated.

Multiplier Type	Number of Components							
	INV	AND	NAND	OR2	NOR	OR3		
CUA	-	n^2	$-n-1$	-	-	-		
PUA	-	$(n^2+3n-2)/2$	-	$n-1$	-	$n-2$		
CUT	-	n^2	-	$n-1$	-	-		
PUT	-	n^2	-	$(n^2+n-4)/2$	-	-		
CSA	$2n-1$	n^2	-	$n-1$	-	-		
PSA	4	$((n^2+5n+2)/2)$	2	$n+1$	$2n-3$	-		
CST	$2n-1$	n^2	-	$n-1$	-	-		
PST	4	$((n^2+7n+4)/2)$	2	$n+1$	$2n-3$	-		

Table 5.1: Component Counts for n-bit Multipliers with Overflow Detection I.

Multiplier Type	Number of Components							
	XOR	XNOR	HA	FA	CPA			
CUA	-	-	n	n^2-2n	-			
PUA	-	-	$n-1$	$(n^2-3n+2)/2$	-			
CUT	-	-	$n-1$	n^2-4n+3	$2n-2$			
PUT	-	-	$n-2$	$(n^2-5n+6)/2$	$n-1$			
CSA	n	-	n	n^2-2n	-			
PSA	$n-3$	$2n-4$	$n-3$	$(n^2-5n+6)/2$	-			
CST	-	n	$n-1$	n^2-4n+3	$2n-2$			
PST	$2n-5$	$2n-4$	$n-2$	$(n^2-5n+6)/2$	n			

Table 5.2: Component Counts for n-bit Multipliers with Overflow Detection II.

Multiplier Type	Number of Components on Worst Case Delay Path						
	INV	AND	OR2	XOR	HA	FA	CPA
CUA	-	1	2	-	2	$2n-4$	-
PUA	-	1	1	-	1	$n-2$	-
CUT	-	1	$\lceil log_2(n) \rceil$	-	-	s	$2n-2$
PUT	-	1	1	-	-	s	$n-1$
CSA	1	1	2	1	2	$2n-4$	-
PSA		2	8	-	1	$n-3$	-
CST	1	1	$\lceil log_2(n) \rceil$	1	-	s	$2n-2$
PST	-	-	1	1	-	$s+1$	$n+1$

Table 5.3: Worst Case Delay for n-bit Multipliers with Overflow Detection.

	Conventional		Proposed		Reduction	
n	Area	Delay	Area	Delay	Area	Delay
8	662	10.70	330	6.19	50%	42%
16	2862	23.02	1366	13.54	52%	41%
24	6598	35.34	3106	20.92	53%	41%
32	11870	47.66	5550	28.26	53%	41%

Table 5.4: Unsigned Array Multipliers with Overflow Detection.

It is possible to reduce the amount of logic required to implement the detection circuit even further. The proposed method uses a straight forward implementation of the logic equations and structures presented in the previous chapters. Synthesis tools are used to further optimize the design. Consequently, the values shown in Table 5.1, 5.2, and Table 5.3 should be considered to be worst case values, before further optimization is performed.

Gate level VHDL code for various sizes of array and Dadda tree multipliers were generated for the conventional and proposed methods for overflow detection. The VHDL code was synthesized and optimized for area using LSI Logic's 0.6 micron LCA300K gate array library and the Leonardo Synthesis tool from Exemplar logic. The synthesis tool was set to a nominal operating voltage of 5.0 volts and a temperature of 25° C. Area estimates are reported in equivalent gates and delay estimates are reported in nanoseconds.

Table 5.4 gives area and delay estimates for unsigned array multiplier. Compared to the multipliers that use conventional overflow detection, the proposed multipliers have between 50% and 53% less area and between 41% and 42% less delay. These

	Conventional		Proposed		Reduction	
n	Area	Delay	Area	Delay	Area	Delay
8	821	8.07	434	6.21	47%	23%
16	2905	10.53	1546	7.85	47%	25%
24	6270	13.57	3344	9.71	47%	28%
32	10918	14.98	5818	11.17	47%	25%

Table 5.5: Unsigned Dadda Tree Multipliers with Overflow Detection.

	Conventional		Proposed		Reduction	
n	Area	Delay	Area	Delay	Area	Delay
8	673	11.03	415	6.45	38%	41%
16	2881	23.35	1617	12.69	44%	48%
24	6625	35.67	3578	18.72	46%	48%
32	11905	47.99	6305	24.88	47%	48%

Table 5.6: Signed Array Multipliers with Overflow Detection.

gains are mainly due to the reductions in the area and the delay of FAs used to generate the n most significant product bits.

Table 5.5 gives area and delay estimates for unsigned Dadda tree multipliers. Compared to multipliers that use conventional overflow detection method, multipliers that use the proposed method have approximately 47% less area and between 23% and 28% less delay. These improvements are due to the reducing the number of FAs and reducing the size of the final carry-propagate adder from $(2n-2)$ to $(n-1)$.

Table 5.6 gives area and delay estimates for two's complement array multipliers. Compared to the multipliers that use the conventional method, multipliers that use the proposed method require between 38% and 47% less area and between 41% and 48% less delay.

	Conventional		Proposed		Reduction	
n	Area	Delay	Area	Delay	Area	Delay
8	837	8.59	547	5.81	35%	32%
16	2925	11.12	1764	8.30	40%	25%
24	6298	13.81	3618	10.24	43%	26%
32	10948	15.24	6115	11.62	44%	24%

Table 5.7: Signed Dadda Tree Multipliers with Overflow Detection.

Table 5.7 gives area and delay estimates for two's complement Dadda tree multipliers. Compared to the multipliers that use the conventional method, multipliers that use the proposed method have between 35% and 44% less area and between 24% to 32% less delay.

Chapter 6

Conclusions and Future Research

6.1 Conclusions

The overflow detection and saturation methods presented in this thesis significantly reduce the area and delay of array and tree multipliers. For the multiplier sizes examined, the area is reduced by about 50% for unsigned multipliers when compared with conventional methods. The proposed methods also do not change the regularity of the multiplier structure. For the two's complement multipliers, the proposed methods are completely independent of the multipliers' internal structure. This feature provides designers increased flexibility, since they can add overflow detection logic without effecting their original design. Reduction in the multiplier hardware will also lead to reduced power dissipation. The proposed methods reduce the delay of array multipliers by about 40% to 50%.

6.2 Future Research

This thesis separately presented overflow detection and saturation methods for unsigned and two's complement parallel multipliers. An important next step is to develop a single multiplier structure that can perform both unsigned and two's complement integer multiplication with overflow detection or saturation based on an input control signal. Another area for future research is to investigate techniques for further reducing the area for overflow detection in multiplier trees, without significantly impacting the delay. This research may be able to take advantage of a hybrid structure that has less delay than linear overflow detection structures and less area than overflow detection trees. Another research area is to investigate reductions in power dissipation due to the proposed techniques. It is anticipated that a significant reduction in power dissipation can be achieved due to the reduction in multiplier hardware. Methods similar to the proposed methods can be also used for other arithmetic operations that needs overflow detection, such as multiply-accumulate and squaring.

Bibliography

[1] L. Dadda," Some Schemes for Parellel Multipliers," *Alta Frequenza 34*, pp.346-356, March 1995.

[2] C.S. Wallace, "Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers*, EC-13, pp. 14-17, 1964.

[3] S.D. Pezaris, "A 40 ns 17-bit Array Multiplier," I*EEE Transactions on Computers*, vol.20, pp. 442-447, 1971.

[4] K. Bickerstaff, M.J. Schulte and E.E. Swartzlander, Jr., "Parallel Reduced Area Multipliers," *Journal of VLSI Signal Processing*, vol. 9, pp.181-192, 1995.

[5] M.E. Robinson and E. Swartzlander, Jr., "A reduction Scheme to Optimize The Wallace Multiplier," *Proceedings of the IEEE International Conference on Computer Design*,pp. 122-127, 1998.

[6] C.R. Baugh and B.A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 22, pp. 1045-1047,

December 1973.

[7] P.E. Blankenship,"Comments on A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 23, p. 1327, 1974.

[8] J.A. Gibson and R.W. Gibbard, "Synthesis and Comparision of Two's Complement Parallel Multipliers," *IEEE Transactions on Computers*, vol. 24, pp. 1020-1027, October 1975.

[9] J.L. Hennesey, "*Computer Architecture a Quantative Approach, Second Edition,*" San Fransisco, Morgan Kauffman, 1996.

[10] N. Yadav, J. Glossner, and M.J. Schulte, "Parallel Saturating Fractional Arithmetic Units," *Proceedings of Nineth Great Lakes Symposium on VLSI,* pp. 214-217, March 1999.

[11] W.R. Hedeman III, "Fixed-point Overflow Exception Detection," *IBM Technical Disclosure Bulletin,* vol. 24, pp. 3126-3127, December 1981.

[12] R.F. Woods, G. Floyd, K. Wood, R .Evans, J.V. McCanny "Programmable High-performance IIR Filter Chip," *IEE Proceedings: Circuits, Devices and Systems*, vol. 142, pp.179-185, June 1995.

[13] D.G. East and J.W. Moore, "Overflow Indication In Two's Complement Arithmetic," *IBM technical Disclosure Bulletin,* vol.19, pp. 3135-3136, January 1977.

[14] P.D. Pai and A. Tran, "Overflow Detection in Multioperand Addition," *International Journal of Electronics,* vol. 73, pp. 461-469, September 1992.

[15] B. Parhami, "Zero, Sign, and Overflow Detection Schemes For Generalized Signed Arithmetic" *Twenty-Second Asilomar Confrence on Signals, Systems, and Computers,* vol. 2, pp. 636-639, 1988.

[16] A.D. Booth, "A Signed Binary Multiplication Technique," *Quaterly Journal of Mechanics and Applied Mathematics,* vol. 4, pp. 236-240, 1951.

Vita

Mustafa Gök was born in Adana, Turkey on the 21st of November, 1972. He is the oldest son of Şener Gök and Hülya Gök. After graduating from Adana Anatolian High School in 1990, he started his undergraduate study in the Department of Electronics Engineering of Istanbul University, Istanbul, Turkey and obtained his B.S. degree in 1995. He joined Lehigh University in 1998.

END OF TITLE